Secured Resource Allocation using Virtual Machines Dynamically in Cloud Computing Environment

Mr. R. Aravind[1] Mrs. A. Mary Joycy[2]
[1]M. E. Student [2]Assistant Professor
[1,2] RatnaVel Subramaniam College of Engineering & Technology

Abstract—Cloud computing allows business customers to scale up and down their resource usage based on needs. Many of the touted gains in the cloud model come from resource multiplexing through virtualization technology. In this paper, we present a system that uses virtualization technology to allocate data center resources dynamically based on application demands and support green computing by optimizing the number of servers in use. We introduce the concept of "skewness" to measure the unevenness in the multidimensional resource utilization of a server. By reducing skewness, we can combine various types of workloads nicely and improve the overall utilization of server resources. We develop a set of heuristics that prevent overload in the system effectively while saving energy used. Trace driven simulation and experiment analysis and results demonstrate that our algorithm achieves good performance.

Keywords*:* Cloud computing, Resource Management, Virtualization, Green computing.

I. INTRODUCTION

The elasticity and the lack of upfront capital investment offered by cloud computing is appealing to many businesses. There is a lot of discussion on the benefits and costs of the cloud model and on how to move legacy applications onto the cloud platform. Here we study a different problem: how can a cloud service provider best multiplex its virtual resources onto the physical hardware? This is important because much of the touted gains in the cloud model come from such multiplexing. Studies have found that servers in many existing data centers are often severely underutilized due to overprovisioning for the peak demand. The cloud model is expected to make such practice unnecessary by offering automatic scale up and down in response to load variation. Besides reducing the hardware cost, it also saves on electricity which contributes to a significant portion of the operational expenses in large data centers. Virtual machine monitors (Shortly Virtual machines) like Xen provide a mechanism for mapping virtual machines to physical resources. This mapping is largely hidden from the cloud users. Users with the Amazon EC2 service, for example, do not know where their Virtual machine instances run. It is up to the cloud provider to make sure the underlying physical machines (Shortly PMs) have sufficient resources to meet their needs. Virtual machine live migration technology makes it possible to change the mapping between Virtual machines and PMs While applications are running. However, a policy issue remains as how to decide the mapping adaptively so that the resource demands of Virtual machines are met while the number of PMs used is minimized. This is challenging when the resource needs of Virtual machines are heterogeneous due to the diverse set of applications they run and vary with time as the workloads grow and shrink. The capacity of PMs can also be heterogeneous because multiple generations of hardware coexist in a data center. We aim to achieve two goals in our algorithm:

i) Overload avoidance. The capacity of a PM should be sufficient to satisfy the resource needs of all Virtual machines running on it. Otherwise, the PM is overloaded and can lead to degraded performance of its Virtual machines.

ii) Green computing. The number of PMs used should be minimized as long as they can still satisfy the needs of all Virtual machines. Idle PMs can be turned off to save energy.

There is an inherent tradeoff between the two goals in the face of changing resource needs of Virtual machines. For overload avoidance, we should keep the utilization of PMs Low to reduce the possibility of overload in case the resource needs of Virtual machines increase later. For green computing, we should keep the utilization of PMs reasonably high to make efficient use of their energy. In this paper, we present the design and implementation of an automated resource management system that achieves a good balance between the two goals.

II. SYSTEM MODEL

Each Virtual machine in domain U encapsulates one or more applications such as Web server, remote desktop, DNS, Mail, Map/ Reduce, etc. We assume all PMs Share backend storage. The multiplexing of Virtual machines to PMs is managed using the Usher framework. The main logic of our system is implemented as a set of plug-ins to U shear. Each node runs an Usher local node manager (LNM) on domain 0 which collects the usage statistics of resources for each Virtual machine on that node. The CPU and network usage can be calculated by monitoring the scheduling events in Xen. The memory usage within a Virtual machine, however, is not visible to hypervisor.

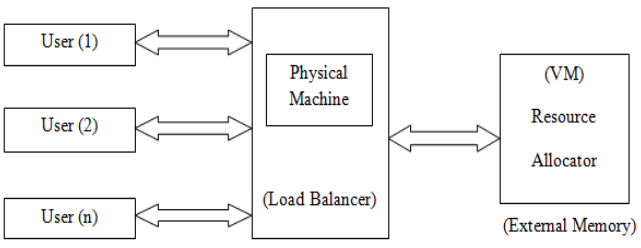

Fig. 1: System Architecture

We use the random page sampling technique as in the Virtual machine ware ESX Server. The statistics collected at each PM are forwarded to the Usher central controller (Usher CTRL) where our Virtual machine scheduler runs. The Virtual machine Scheduler is invoked periodically and receives from the LNM the resource

demand history of Virtual machines, the capacity and the load history of PMs, and the current layout of Virtual machines on PMs.

A. Forecasting Future Need of Resources

We need to predict the future resource needs of Virtual machines. As said earlier, our focus is on Internet applications. One solution is to look inside a Virtual machine for application level statistics, e.g., by parsing logs of pending requests. Doing so requires modification of the Virtual machine which may not always be possible. Instead, we make our prediction based on the past external behaviors of Virtual machines.

$$E(t) = \alpha * E(t-1) + (1-\alpha) * O(t), 0 \leq \alpha \leq 1,$$

Our first attempt was to calculate an exponentially weighted moving average (EWMA) using a TCP-like scheme are the estimated and the observed load at time t, respectively.

We use the EWMA formula to predict the CPU load on the DNS server in our university. We measure the load every minute and predict the load in the next minute. Fig. 2 shows the results for 0:7. Each dot in the figure is an observed value and the curve represents the predicted values. Visually, the curve cuts through the middle of the dots which indicates a fairly accurate prediction. This is also verified by the statistics in Table 1. The parameters in the parenthesis are the values. W is the length of the measurement window (explained later). As we can see, the prediction is fairly accurate with roughly equal percentage of higher and lower values. Although seemingly satisfactory, this formula does not capture the rising trends of resource usage. For example, when we see a sequence of 10; 20; 30, and 40, it is reasonable to predict the next value to be 50. Unfortunately, when it is between 0 and 1, the predicted value is always between the historic value and the observed one.

III. PROPOSED (SKEWNESS) ALGORITHM

We introduce the concept of skewness to quantify the unevenness in the utilization of multiple resources on a server. Let n be the number of resources we consider and *ri* be the utilization of the *i'*th with resource. We define the resource skewness of a server p as

$$skewness(p) = \sqrt{\sum_{i=1}^{n} \left(\frac{r_i}{\overline{r}} - 1\right)^2},$$

(1) median error
(2) High Error
(3) low error

Table. 1: Load Prediction Algorithms

A. Hot and Cold Acnes

Our algorithm executes periodically to evaluate the resource allocation status based on the predicted future resource demands of Virtual machines. We define a server as a hot spot if the utilization of any of its resources is above a hot threshold. This indicates that the server is overloaded and hence some Virtual machines running on it should be migrated away. We define the temperature of a hot spot p as the square sum of its resource utilization beyond the hot threshold:

$$\text{Temperature (p)} = \sum (r - r_t)^2$$

Where R is the set of overloaded resources in server p and *rt* is the hot threshold for resource r. The temperature of a hot spot reflects its degree of overload. If a server is not a hot spot, its temperature is '0'. We define a server as a cold spot if the utilizations of all its resources are below a cold threshold. This indicates that the server is mostly idle and a potential candidate to turn off to save energy. However, we do so only when the average resource utilization of all actively used servers in the system is below a green computing threshold.

A server is actively used if it has at least one Virtual machine running. Otherwise, it is inactive. Finally, we define the warm threshold to be a level of resource utilization that is sufficiently high to justify having the server running but not as high as to risk becoming a hot spot in the face of temporary fluctuation of application resource demands. Different types of resources can have different thresholds. 90 % or its memory usage is above 80 %.

B. Green Figuring

When the resource utilization of active servers is too low, some of them can be turned off to save energy. This is handled in our green computing algorithm. The challenge here is to reduce the number of active servers during low load without sacrificing performance either now or in the future. We need to avoid oscillation in the system.

Our green computing algorithm is invoked when the average utilizations of all resources on active servers are below the green computing threshold. We sort the list of cold spots in the system based on the ascending order of their memory size. Since we need to migrate away all its Virtual machine before we can shut down an underutilized server, we define the memory size of a cold spot as the aggregate memory size of all Virtual machine running on it. Recall that our model assumes all Virtual machine connect to share back end storage. Hence, the cost of a Virtual machine live migration is determined mostly by its memory footprint.

IV. LIMITATIONS

We evaluate the performance of our algorithm using trace driven simulation. Note that our simulation uses the same code base for the algorithm as the real implementation in the experiments. This ensures the fidelity of our simulation results. The raw traces are preprocessed into "Usher" format so that the simulator can read them. We collected the traces from a variety of sources:

(1) Web Info Mall. The largest online Web archive in China with more than three billion archived Web pages.
(2) Real Course. The largest online distance learning system in China with servers distributed across thirteen major cities.
(3) Amazing Store. The largest Point to Point storage system in China. We also collected traces from servers and desktop computers in our university including one of our mail servers, the central DNS server, and desktops in our department. We post processed the traces based on days

collected and use random sampling and linear combination of the data sets to generate the workloads needed. All simulation in this section uses the real trace workload unless otherwise specified.

A. Result of Beginnings on APMs

We first evaluate the effect of the various thresholds used in our algorithm. We simulate a system with Thousand PMs and Thousand Virtual machines. We use random Virtual machine to PM mapping in the initial layout. The scheduler is invoked once per minute. The x axis is the time of the day starting at 8 am. The y-axis is overloaded with two meanings: the percentage of the load or the percentage of APMs in the system. Recall that a PM is active if it has at least one Virtual machine running. As can be seen from the figure, the CPU load demonstrates diurnal patterns which decrease substantially after midnight.

The memory consumption is fairly stable over the time. The network utilization stays very low. Our algorithm can be made more or less aggressive in its migration decision by tuning the thresholds. The figure shows that lower hot thresholds cause more aggressive migrations to mitigate hot spots in the system and increases the number of APMs, and higher cold and green computing thresholds cause more aggressive consolidation which leads to a smaller number of APMs. With the default thresholds in Table 2, the percentage of APMs in our algorithm follows the load pattern closely.

B. Scalability of the Set of rules

We evaluate the scalability of our algorithm by varying the number of Virtual machines in the simulation between two hundred and thousand four hundred. The speed of increase is between linear and quadratic. We break down the decision time into two parts: hot spot mitigation and green computing. We find that hot spot mitigation contributes more to the decision time. We also find that the decision time for the synthetic workload is higher than that for the real trace due to the large variation in the synthetic workload. With one forty PMs and thousand four hundred Virtual machines, the decision time is about I.III seconds for the synthetic workload and 0.2 second for the real trace. The number of migrations is small and increases roughly linearly with the system size. We find that hot spot contributes more to the number of migrations. We also find that the number of migrations in the synthetic workload is higher than that in the real trace. With hundred forty PMs and thousand four hundred Virtual machines, on average each run of our algorithm incurs about three migrations in the whole system for the synthetic workload and only migrations for the real trace.

C. Result of Weight Forecast

We compare the execution of our algorithm with and without load prediction. When load prediction is disabled, the algorithm simply uses the last observed load in its decision making. That load prediction significantly reduces the average number of hot spots in the system during a decision run. Notably, prediction prevents Over 46 % hot spots in the simulation with thousand four hundred Virtual machines.

This demonstrates its high effectiveness in preventing server overload proactively. Without prediction, the algorithm tries to consolidate a PM as soon as its load drops below the threshold. With prediction, the algorithm correctly foresees that the load of the PM will increase above the threshold shortly and hence takes no action. This leaves the PM in the "cold spot" state for a while. However, it also reduces placement churns by avoiding unnecessary migrations due to temporary load fluctuation. Consequently, the number of migrations in the system with load prediction is smaller than that without prediction as shown in Fig. 6c. We can adjust the conservativeness of load prediction by tuning its parameters, but the current configuration largely serves our purpose. The only downside of having more cold spots in the system is that it may increase the number of APMs.

V. CORRELATED WORKS

Resource Allocation at the Application Level Automatic scaling of Web applications was previously studied for data center environments. Each server has replicas of all web applications running in the system. The dispatch algorithm in a frontend L7-switch makes sure requests are reasonably served while minimizing the number of underutilized servers.

Work uses network flow algorithms to allocate the load of an application among its running instances. For connection oriented Internet services like Windows Live Messenger, work presents an integrated approach for load dispatching and server provisioning. All works above do not use virtual machines and require the applications be structured in a multitier architecture with load balancing provided through a front-end dispatcher. In contrast, our work targets Amazon EC2-style environment where it places no restriction on what and how applications are constructed inside the Virtual machines.

A Virtual machine is treated like a black box. Resource management is done only at the granularity of whole Virtual machines. Map Reduce is another type of popular Cloud service where data locality is the key to its performance. Quincy adopts min cost flow model in task scheduling to maximize data locality while keeping fairness among different jobs. The "Delay Scheduling" algorithm trades execution time for data locality. Work assigns dynamic priorities to jobs and users to facilitate resource allocation. Resource Allocation by Live Virtual machine Migration Virtual machine live migration is a widely used technique for dynamic resource allocation in a virtualized environment. Our work also belongs to this category.

Sandpiper combines multidimensional load information into a single Volume metric. It sorts the list of PMs based on their volumes and the Virtual machines in each PM in their volume-to-size ratio (shortly VSR). This unfortunately abstracts away critical information needed when making the migration decision. It then considers the PMs and the Virtual machines in the presorted order. We give a concrete example in Section 1 of the supplementary file, which is available online, where their algorithm selects the wrong Virtual machine to migrate away during overload and fails to mitigate the hot spot.

We also compare our algorithm and theirs in real experiment. The results are analyzed in Section 5 of the supplementary file, which is available online, to show how they behave differently. In addition, their work has no support Virtual machine for green computing and differs from ours in many other aspects such as load prediction. The HARMONY system applies virtualization technology across multiple resource layers. It uses Virtual machine and data migration to mitigate hot spots not just on the servers, but also on network devices and the storage nodes as well. It introduces the Extended Vector Product (Shortly EVP) as an indicator of imbalance in resource utilization. Their load balancing algorithm is a variant of the Toyoda method for multidimensional knapsack problem. Unlike our system, their system does not support green computing and load prediction is left as future work. In Section five of the supplementary file, which is available online, we analyze the phenomenon that Vector Dot behaves differently compared with our work and point out the reason why our algorithm can utilize residual resources better. Dynamic placement of virtual servers to minimize SLA violations is studied.

They model it as a bin packing problem and use the well-known first-fit approximation algorithm to calculate the Virtual machine to PM layout periodically. That algorithm, however, is designed mostly for offline use. It is likely to incur a large number of migrations when applied in online environment where the resource needs of Virtual machines change dynamically.

A. Green Computing

Many efforts have been made to curtail energy consumption in data centers. Hardware-based approaches include novel thermal design for lower cooling power, or adopting power proportional and low-power hardware. Work uses dynamic voltage and frequency scaling (shortly DVFS) to adjust CPU power according to its load. We do not use DVFS for green computing, as explained in Section 7 of the supplementary file.

Powernap resorts to new hardware technologies such as solid state disk (shortly SSD) and Self-Refresh DRAM to implement rapid transition (>then 1ms) between Full operation and low power state, so that it can "take a nap" in short idle intervals. When a server goes to sleep notifies an embedded system residing on a special designed NIC to delegate the main operating system. It gives the illusion that the server is always active. Our work belongs to the category of pure-software low-cost solutions initiate's virtual machines on a dedicated server as delegate, instead of depending on a special NIC. Invents "partial Virtual machine migration," a variance of live Virtual machine migration, which only migrates away necessary working set while leaving infrequently used data behind.

VI. CONCLUSION

We have presented the design, implementation, and evaluation of a resource management system for cloud computing services. Our system multiplexes virtual to physical resources adaptively based on the changing demand.

We use the skewness metric to combine VMs with different resource characteristics appropriately so that the capacities of servers are well utilized. Our algorithm achieves both overload avoidance and green computing for systems with multi resource constraints.

REFERENCES

[1] Y. Agarwal, S. Hodges, R. Chandra, J. Scott, P. Bahl, and R. Gupta, "Somniloquy: Augmenting Network Interfaces to Reduce Pc Energy Usage," 2009.
[2] T. Das, P. Padala, V.N. Padmanabhan, R. Ramjee, and K.G. Shin, "Litegreen: Saving Energy in Networked Desktops Using Virtualization", 2010.
[3] Y. Agarwal, S. Savage, and R. Gupta, "Sleep server: A Software- Only Approach for Reducing the Energy Consumption of PCS within Enterprise Environments," 2010.
[4] N. Bila, E. d. Lara, K. Joshi, H. A. Lagar-Cavilla, M. Hiltunen, and M. Satyanarayanan, "Jettison: Efficient Idle Desktop Consolidation with Partial VM Migration," Proc. ACM European Conf. Computer Systems, 2012.

